C# Fundamentals

Part 4

Database

ADO.Net

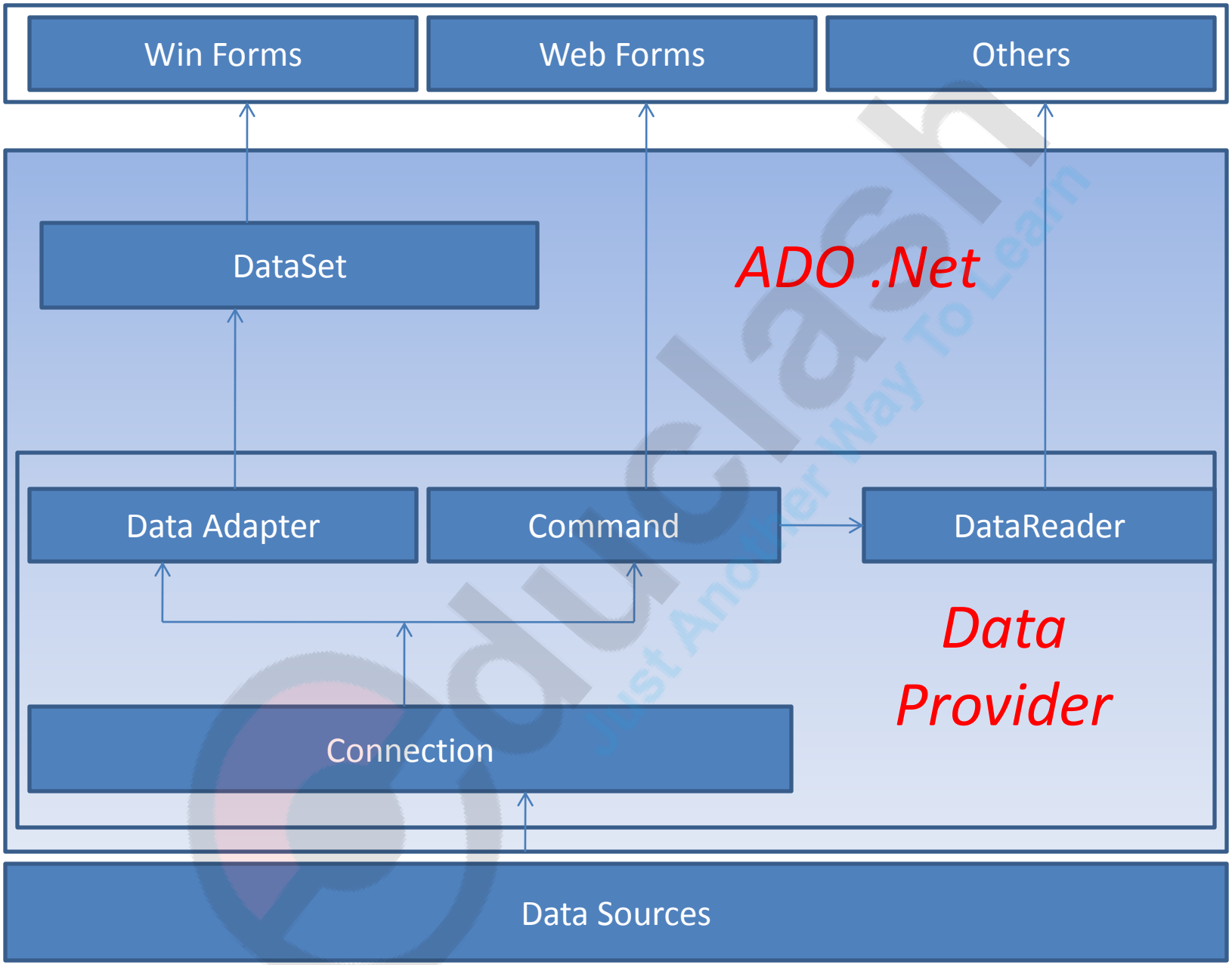
ADO.NET separates data access from data manipulation into discrete components.

ADO.NET includes .NET Framework data providers for connecting to a database, executing commands, and retrieving results.

The ADO.NET classes are found in System.Data.dll

ADO.Net Architecture

The two main components of ADO.NET 3.0 for accessing and manipulating data are the .NET Framework data providers(Connected Architecture) and the DataSet (Disconnected Architecture).


Win Forms
Web Forms
Others
DataSet
ADO .Net
Data Adapter
Command
DataReader
Data Provider
Connection
Data Sources

.NET Framework Data Providers

.NET Framework Data Providers

The .NET Framework Data Providers are components that have been explicitly designed for data manipulation and fast, forward-only, read-only access to data.

.NET Framework Data Providers

The following table lists the data providers that are included in the .NET Framework.

.NET Framework data provider	Description
.NET Framework Data Provider for SQL Server	Provides data access for Microsoft SQL Server. Uses the System.Data.SqlClient namespace.
.NET Framework Data Provider for OLE DB	For data sources exposed by using OLE DB. Uses the System.Data.OleDb namespace.
.NET Framework Data Provider for ODBC	For data sources exposed by using ODBC. Uses the System.Data.Odbc namespace.
.NET Framework Data Provider for Oracle	For Oracle data sources. The .NET Framework Data Provider for Oracle supports Oracle client software version 8.1.7 and later, and uses the System.Data.OracleClient namespace.
EntityClient Provider	Provides data access for Entity Data Model (EDM) applications. Uses the System.Data.EntityClient namespace.
.NET Framework Data Provider for SQL Server Compact 4.0.	Provides data access for Microsoft SQL Server Compact 4.0. Uses the System.Data.SqlServerCe namespace.

.NET Framework Data Providers

Core classes of .NET Framework Data Providers:

- The **Connection** object provides connectivity to a data source.
- The **Command** object enables access to database commands to return data, modify data, run stored procedures, and send or retrieve parameter information.
- The **DataReader** provides a high-performance stream of data from the data source.
- **DataAdapter** provides the bridge between the **DataSet** object and the data source.
- **DataAdapter** uses **Command** objects to execute SQL commands at the data source to both load the **DataSet** with data and reconcile changes that were made to the data in the **DataSet** back to the data source.

The DataSet

• The ADO.NET **DataSet** is explicitly designed for data access independent of any data source.

• Hence it can be used with multiple and differing data sources, it can be used with XML data, or used to manage data local to the application.

• The **DataSet** contains a collection of one or more DataTable objects consisting of rows and columns of data, and also primary key, foreign key, constraint, and relation information about the data in the **DataTable** objects.

DataSet

DataSet:

The ADO.NET DataSet is a memory-resident representation of data.

A DataSet represents a complete set of data including the tables that contain, order, and constrain the data, as well as the relationships between the tables.

There are several ways of working with a DataSet, which can be applied indepen dently or in combination. You can:

- Programmatically create a DataTable, DataRelation, and Constraint within a DataSet and populate the tables with data.
- Populate the DataSet with tables of data from an existing relational data source using a **DataAdapter**.
- Load and persist the DataSet contents using XML. For more information, see Using XML in a DataSet.

DataTable:

The DataTable is a central object in the ADO.NET library. Other objects that use the DataTable include the DataSet and the DataView.

If you are creating a DataTable programmatically, you must first define its schema by adding DataColumn objects to the DataColumnCollection.

You create DataColumn objects within a table by using the DataColumn objects he DataColumn constructor,

or by calling the Add method of the Columns property of the table.

The **Add** method accepts optional **ColumnName**, **DataType**,
and **Expression** arguments and creates a new **DataColumn** as a member of the collection.

The following example adds four columns to a DataTable.

```
DataTable workTable = new DataTable("Customers");

DataColumn workCol = workTable.Columns.Add("CustID", typeof(Int32));
workCol.AllowDBNull = false;
workCol.Unique = true;

workTable.Columns.Add("CustLName", typeof(String));
workTable.Columns.Add("CustFName", typeof(String));
workTable.Columns.Add("Purchases", typeof(Double));
```

Defining Primary Keys

A database table commonly has a column or group of columns that uniquely identifies each row in the table. This identifying column or group of columns is called the primary key.

When you identify a single DataColumn as the PrimaryKey for a DataTable, the table automatically sets the AllowDBNull property of the column to **false** and the Unique property to **true**.

For multiple-column primary keys, only the **AllowDBNull** property is automatically set to **false**.

The **PrimaryKey** property of a DataTable receives as its value an array of one or more **DataColumn** objects, as shown in the following examples.

The first example defines a single column as the primary key.

```
workTable.PrimaryKey = new DataColumn[] {workTable.Columns["CustID"]};

// Or

DataColumn[] columns = new DataColumn[1];
columns[0] = workTable.Columns["CustID"];
workTable.PrimaryKey = columns;
```

The following example defines two columns as a primary key

```
workTable.PrimaryKey = new DataColumn[] {workTable.Columns["CustLName"],
                                         workTable.Columns["CustFName"]};

// Or

DataColumn[] keyColumn = new DataColumn[2];
keyColumn[0] = workTable.Columns["CustLName"];
keyColumn[1] = workTable.Columns["CustFName"];
workTable.PrimaryKey = keyColumn;
```

Adding rows to table

To add rows to a DataTable, you must first use the NewRow method, which return a new DataRow object.

The NewRow method returns a row with the schema of the DataTable.

Adding Data to a DataTable

After you create a DataTable and define its structure using columns and constraints, you can add new rows of data to the table.

To add a new row, declare a new variable as type DataRow.

A new **DataRow** object is returned when you call the NewRow method.

The **DataTable** then creates the **DataRow** object based on the structure of the table

The following example demonstrates how to create a new row by calling the **NewRow** method.

```
DataRow workRow = workTable.NewRow();
```

You then can manipulate the newly added row using an index or the column name, as shown in the following example.

```
workRow["CustLName"] = "Smith";
workRow[1] = "Smith";
```

After data is inserted into the new row, the **Add** method is used to add the row to the DataRowCollection, shown in the following code.

```
workTable.Rows.Add(workRow);
```

The following example adds 10 rows to the newly created **Customers** table.

```
DataRow workRow;

for (int i = 0; i <= 9; i++)
{
  workRow = workTable.NewRow();
  workRow[0] = i;
  workRow[1] = "CustName" + i.ToString();
  workTable.Rows.Add(workRow);
}
```

The following example creates a DataTable, adds two DataColumn objects that determine the table's schema.

Creates several new DataRow objects using the NewRow method.

Those DataRow objects are then added to the DataRowCollection using the Add method.

```
private void MakeDataTableAndDisplay()
{
    // Create new DataTable and DataSource objects.
    DataTable table = new DataTable();

    // Declare DataColumn and DataRow variables.
    DataColumn column;
    DataRow row;
    DataView view;

    // Create new DataColumn, set DataType, ColumnName and add to DataTable.
    column = new DataColumn();
    column.DataType = System.Type.GetType("System.Int32");
    column.ColumnName = "id";
    table.Columns.Add(column);

    // Create second column.
    column = new DataColumn();
    column.DataType = Type.GetType("System.String");
    column.ColumnName = "item";
    table.Columns.Add(column);
```

```
        // Create new DataRow objects and add to DataTable.
        for(int i = 0; i < 10; i++)
        {
            row = table.NewRow();
            row["id"] = i;
            row["item"] = "item " + i.ToString();
            table.Rows.Add(row);
        }

        // Create a DataView using the DataTable.
        view = new DataView(table);

        // Set a DataGrid control's DataSource to the DataView.
        dataGrid1.DataSource = view;
    }
```

The maximum number of rows that a DataTable can store is 16,777,216.

DataAdapter class

- **SqlDataAdapter** Class is a part of the C# ADO.NET Data Provider and it resides in the **System.Data.SqlClient** namespace.
- SqlDataAdapter provides the communication between the Dataset and the SQL database.
- We can use SqlDataAdapter Object in combination with Dataset Object.
- DataAdapter provides this combination by mapping Fill() method, which changes the data in the DataSet to match the data in the data source, and
- Update() method, which changes the data in the data source to match the data in the DataSet.

Constructors for DataAdapter:

Name	Description
SqlDataAdapter()	Initializes a new instance of the SqlDataAdapter class.
SqlDataAdapter(SqlCommand)	Initializes a new instance of the SqlDataAdapter class with the specified SqlCommand as the SelectCommand property.
SqlDataAdapter(String, SqlConnection)	Initializes a new instance of the SqlDataAdapter class with a SelectCommand and a SqlConnection object.
SqlDataAdapter(String, String)	Initializes a new instance of the SqlDataAdapter class with a SelectCommand and a connection string.

Major Methods of DataAdapter class:

- Fill(DataSet)Adds or refreshes rows in the DataSet.
- Fill(DataTable)Adds or refreshes rows in a specified DataTable
- Update(DataSet)Updates the values in the database by executing the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the specified DataSet.
- Update(DataSet, String)Updates the values in the database by executing the respective INSERT, UPDATE, or DELETE statements for each inserted, updated, or deleted row in the DataSet with the specified DataTable name.

Connection Oriented Architecture

An Introduction:

The ADO.NET Framework supports two models of Data Access Architecture, Connection Oriented Data Access Architecture and Disconnected Data Access Architecture.

In Connection Oriented Data Access Architecture the application makes a connection to the Data Source and then interact with it through SQL requests using the same connection. In these cases the application stays connected to the database system even when it is not using any Database Operations.

In Connection Oriented Data Access, when you read data from a database by using a DataReader object, an open connection must be maintained between your application and the Data Source.

Classes used in Connected Architecture

• The four Objects from the .Net Framework provide the functionality of Data Providers in ADO.NET.

• They are **Connection** Object,**Command** Object , **DataReader** Object and **DataAdapter** Object.

• The .Net Framework includes mainly three Data Providers for ADO.NET. They are the Microsoft **SQL Server** Data Provider ,**OLEDB** Data Provider and **ODBC** Data provider.

DataReader Class

•**DataReader Object** in ADO.NET is a stream-based , forward-only, read-only retrieval of query results from the Data Sources , which do not update the data.

•The DataReader cannot be created directly from code, they can be created only by calling the **ExecuteReader** method of a Command Object.

• The DataReader Object provides a connection oriented data access to the Data Sources.

• The **Read()** method in the DataReader is used to read the rows from DataReader and it always moves forward to a new valid row, if any row exist .

SqlCommand Class

Represents a Transact-SQL statement or stored procedure to execute against a SQL Server database.

Major Constructors used for SqlCommand:

Name	Description
SqlCommand()	Initializes a new instance of the SqlCommand class.
SqlCommand(String)	Initializes a new instance of the SqlCommand class with the text of the query.
SqlCommand(String, SqlConnection)	Initializes a new instance of the SqlCommand class with the text of the query and a SqlConnection.
SqlCommand(String, SqlConnection, SqlTransaction)	Initializes a new instance of the SqlCommand class with the text of the query, a SqlConnection, and the SqlTransaction.
SqlCommand(String, SqlConnection, SqlTransaction, SqlCommandColumnEncryptionSetting)	Initializes a new instance of the SqlCommand class with specified command text, connection, transaction, and encryption setting.

One of the major property of SqlCommand is **CommandType**

If we want to execute normal SQL Query then use:

```
command.CommandType = CommandType.Text;
```

If we want to execute a stored procedure then use:

```
command.CommandType = CommandType.StoredProcedure;
```

Imp methods of Command Object:

ExecuteReader():

Sends the CommandText to the Connection and builds a DataReader. (Used to get data from data source to datareader)

ExecuteScalar(): (Click For Program)

Executes the query, and returns the first column of the first row in the result set returned by the query. Additional columns or rows are ignored. (Used when the result has a single value for eg Count records in a table OR sum of a column)

ExecuteNonQuery():

You can use the ExecuteNonQuery to perform UPDATE, INSERT, or DELETE operations.

For UPDATE, INSERT, and DELETE statements, the return value is the number of rows affected by the command.